OpenGL: Performance and Optimizations

Session 514

OpenGL: Performance and Optimizations

John Stauffer
OpenGL Engineering Manager

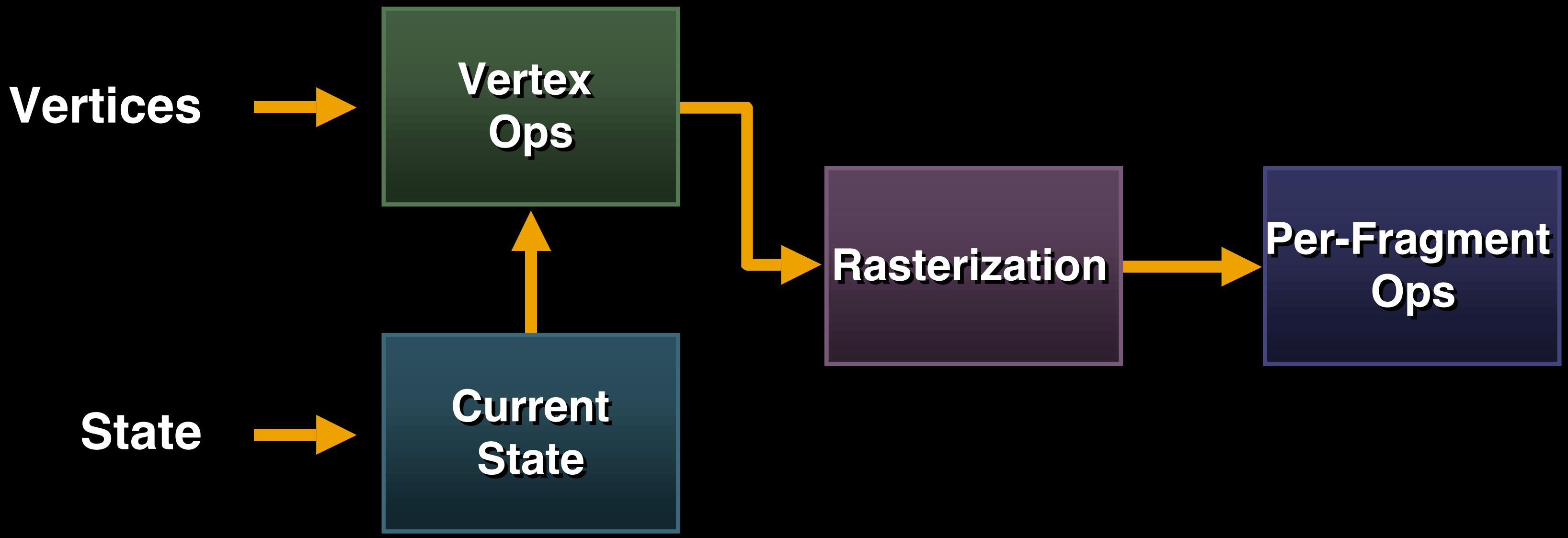
What You’ll Learn

- Basic optimization suggestions
- New optimizations in Jaguar
- OpenGL extensions
- Using threads
- Tools
    - OpenGL Profiler
    - Sampler
- Where to look for help

Basic Optimizations

- Optimization technique is application dependent
    - What is your primary data type?
        - Pixels or vertices
    - Do you changes textures a lot?
    - Do you have complex vertex data sets?
    - Do you spool data from disk in realtime?

OpenGL Pipeline



Basic Optimizations

- ALMOST NEVER call **glFlush**
  - Only call for single buffered contexts
  - Submits the current command buffer
  - Can artificially cause the driver to run out of command buffers
- NEVER call **glFinish**
  - Waits until all pending drawing is finished by the graphics device

Basic Optimizations

- Avoid using **glReadPixels**
  - Can cause an implicit glFinish
  - Only use for saving pixel data
- Avoid using **glDrawPixels**
  - Use a textured Quad instead

Basic Optimizations

- Minimize state changes
  - Textures, enables/disables, etc . . .
  - State changes can account for a large portion on the time spent in OpenGL

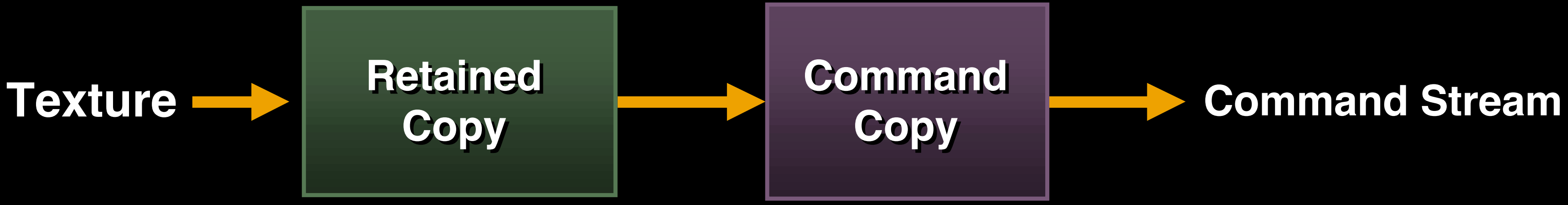
New Optimizations in Jaguar

- **glDrawPixels**
    - Now done with texture Quads
- **glReadPixels**
    - Graphics card now pushes the data to memory instead of CPU reading data across PCI bus
- **glTexSubImage**
    - Now a VRAM to VRAM operation

New Optimizations in Jaguar

- Display Lists
  - Now uses vertex array object extension
  - Recommended for static vertex data
- Vertex Arrays
- Image Processing
  - More Altivec code

OpenGL Texture Pipeline



Basic Texture Optimizations

- Scale texture sizes to hardware
    - Try to avoid texture paging
- Don’t second guess OpenGL texture paging
    - OpenGL uses LRU/MRU paging
    - Paging uses very little CPU
- Use packed pixel types
    - **GL_BGRA,GL_UNSIGNED_SHORT_1_5_5_5_REV**
    - **GL_BGRA, GL_ UNSIGNED_INT_8_8_8_8_REV**

OpenGL Texture Extensions

- **GL_APPLE_client_storage**
    - Eliminates OpenGL storing a texture copy
    - Application texture is directly used by OpenGL
    - Increases texture upload performance
    - Excellent for textures that change a lot
    - Saves memory if application already keeps a copy of the texture

NEW OpenGL Texture Extensions

- **GL_ APPLE_texture_range**
    - Eliminates driver copying texture
    - Dynamically maps pixel data directly into AGP memory
    - **GL_BGRA,GL_UNSIGNED_SHORT_1_5_5_5_REV**
    - **GL_BGRA, GL_ UNSIGNED_INT_8_8_8_8_REV**
    - Use **GL_APPLE_fence** to test for completion

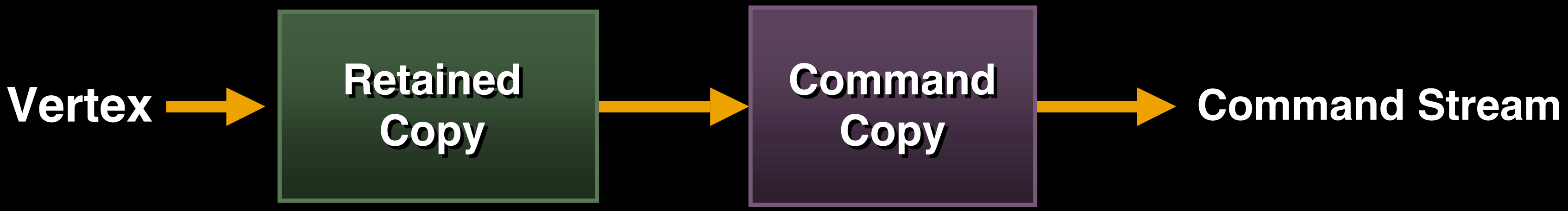
OpenGL Texture Extensions

- **GL_EXT_texture_rectangle**
  - Allows non-power of two textures
  - Fast path through drivers
  - Good for blitting images to the screen
  - Does not allow mipmap filtering
  - Does not allow **GL_REPEAT** wrap mode
  - Texture coords are in pixel space, not normalized space

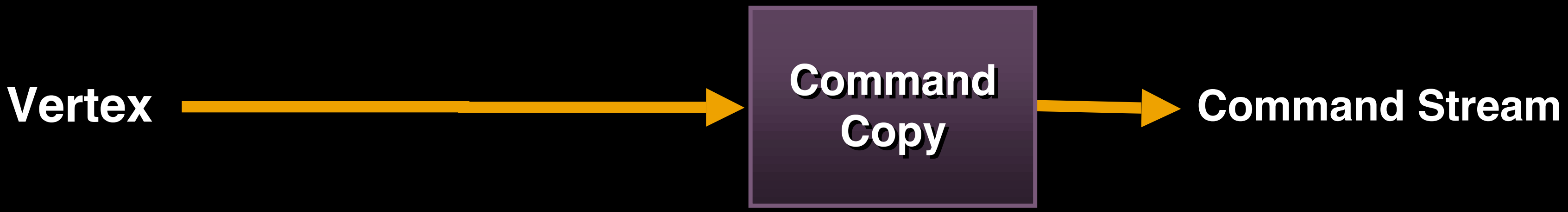
Demo

OpenGL Vertex Pipeline

Immediate Mode



Vertex Array



Basic Vertex Optimizations

- Try to avoid the immediate mode vertex path
- Maximize number of vertices per **glBegin/glEnd**
  - This reduces per function call overhead
- Use efficient primitives
  - Triangle strips, Quad strips
- Use **aglMacros.h/CGLMacros.h**
  - Reduces function call overhead

NEW OpenGL Vertex Extensions

- **GL_APPLE_vertex_array_range**
  - Use if available
  - Most optimized path for TCL cards
  - Eliminates the driver copying the data
  - Dynamically maps vertex data directly into AGP memory
  - Use **GL_APPLE_fence** to test for completion

NEW OpenGL Vertex Extensions

- **GL_APPLE_vertex_array_object**
  - Built on **GL_APPLE_vertex_array_range**
  - Allows for multiple AGP memory regions
  - Offers the same functionality as texture objects

OpenGL Vertex Extensions

- **GL_EXT_compiled_vertex_array**
  - Most optimized path for Non-TCL cards
  - Don’t use if **GL_APPLE_vertex_array_range** is available

Vertex Optimization Example

- Step 1: Starting point

```
for(i = 0; i < num_polys - 2; i++)
{
        glShadeModel(GL_SMOOTH);
        glColor4d (r, g, b, 1.0);
        glBegin (GL_TRIANGLES);
                glVertex4f (v[i][0], v[i][1], v[i][2], 1.0);
                glVertex4f (v[i+1][0], v[i+1][1], v[i+1][2], 1.0);
                glVertex4f (v[i+2][0], v[i+2][1], v[i+2][2], 1.0);
        glEnd ();
}
```

Vertex Optimization Example

- Step 2: Remove static state changes from loops

```
glShadeModel(GL_ SMOOTH);
glColor4d (r, g, b, 1.0);
for(i = 0; i < num_polys - 2; i++)
{
        glBegin (GL_TRIANGLES);
                glVertex4f (v[i][0], v[i][1], v[i][2], 1.0);
                glVertex4f (v[i+1][0], v[i+1][1], v[i+1][2], 1.0);
                glVertex4f (v[i+2][0], v[i+2][1], v[i+2][2], 1.0);
        glEnd ();
}
```

Vertex Optimization Example

- Step 3: Maximize vertices per begin/end

```
  glShadeModel(GL_ SMOOTH);
  glColor4d (r, g, b, 1.0);
  glBegin (GL_TRIANGLES);
          for(i = 0; i < num_polys - 2; i++)
          {
                glVertex4f (v[i][0], v[i][1], v[i][2], 1.0);
                glVertex4f (v[i+1][0], v[i+1][1], v[i+1][2], 1.0);
                glVertex4f (v[i+2][0], v[i+2][1], v[i+2][2], 1.0);
         }
glEnd ();
```

Vertex Optimization Example

- Step 4: Simplify data types

```
glShadeModel(GL_ SMOOTH);
glColor3f (r, g, b);
glBegin (GL_TRIANGLES);
        for(i = 0; i < num_polys - 2; i++)
        {
              glVertex3fv (v[i]);
              glVertex3fv (v[i+1]);
              glVertex3fv (v[i+2]);
        }
glEnd ();
```

Vertex Optimization Example

- Step 5: Simplify primitive types

```
glShadeModel(GL_ SMOOTH);
glColor3f (r, g, b);
glBegin (GL_TRIANGLE_STRIP);
        for(i = 0; i < num_polys; i++)
        {
                glVertex3fv (v[i]);
        }
glEnd ();
```

Vertex Optimization Example

- Step 6: Use vertex arrays

```
glShadeModel(GL_ SMOOTH);
glColor3f (r, g, b);

glVertexPointer (3, GL_FLOAT, 0, v);
glEnableClientState (GL_VERTEX_ARRAY);
glDrawElements (GL_TRIANGLE_STRIP, num_polys,
                GL_UNSIGNED_SHORT, indices);
```

Vertex Optimization Example

- Step 7: Use GL_APPLE_vertex_array_range

```
glShadeModel(GL_ SMOOTH);
glColor3f (r, g, b);

glVertexPointer (3, GL_FLOAT, 0, v);
glEnableClientState (GL_VERTEX_ARRAY);
glVertexArrayRangeAPPLE(sizeof(v), v);
glEnableClientState(GL_VERTEX_ARRAY_RANGE_APPLE);
glFlushVertexArrayRangeAPPLE(sizeof(v), v);
glDrawElements (GL_TRIANGLE_STRIP, num_polys,
                GL_UNSIGNED_SHORT, indices);
```

Demo

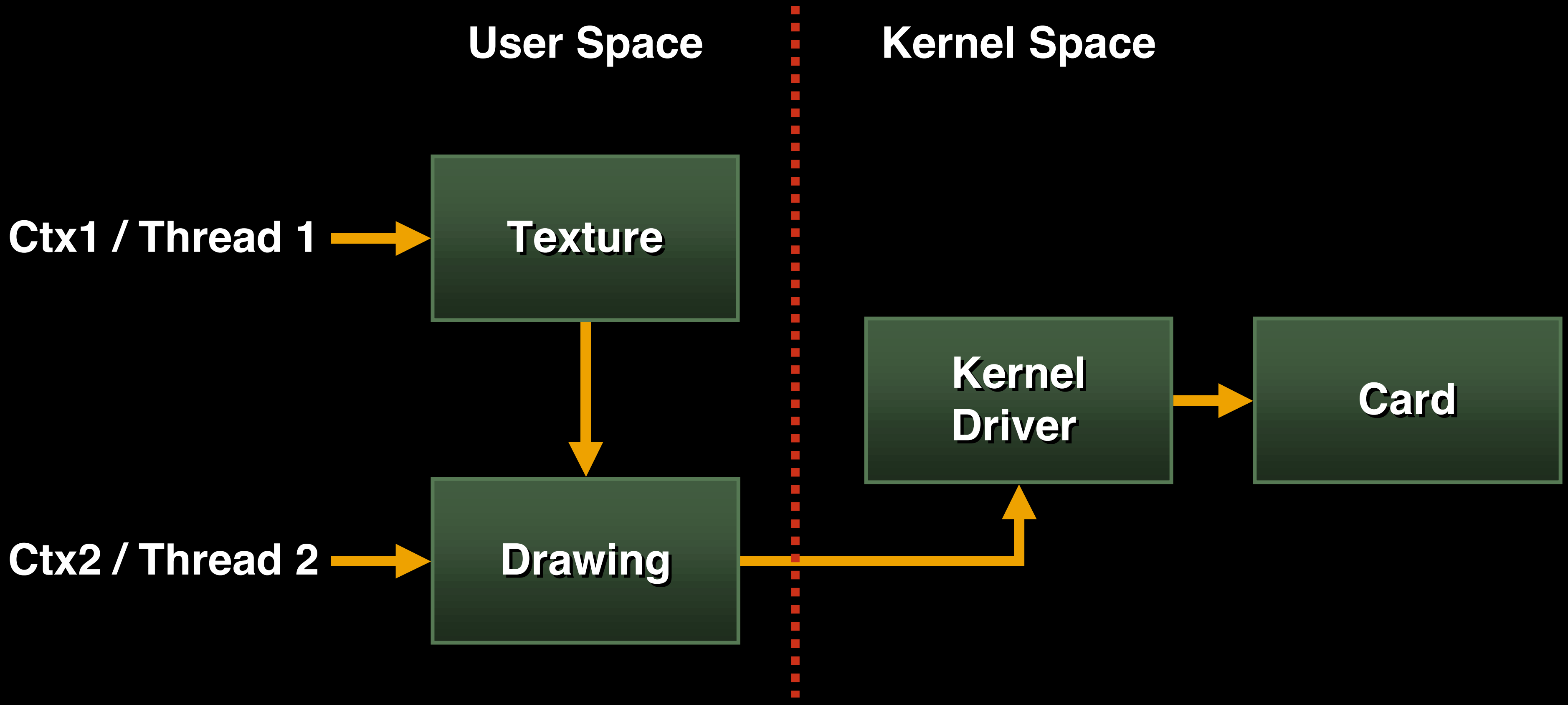
NEW OpenGL Threads

- **GL_APPLE_fence**
    - Provides synchronization tokens
    - Synchronous token query
        - Block until token completed
    - Asynchronous token query
    - Use to synchronize between threads
    - Use to synchronize vertex and texture range operations

Using Threads in OpenGL

- Loading textures in a second thread
    - Allows primary thread to continue uninterrupted drawing
    - Create 2 shared contexts, 1 per thread
- Don’t call an OpenGL context from multiple threads simultaneously without careful synchronization
    - Will crash your application

Threaded Texture Loading



Demo

OpenGL Profiler

- OpenGL function call statistics
- OpenGL function call trace
- Driver statistics
- OpenGL function break points
  - Application call stack
  - OpenGL state
- Noop and profile control for any OpenGL function
- Force buffer flushing after draw function

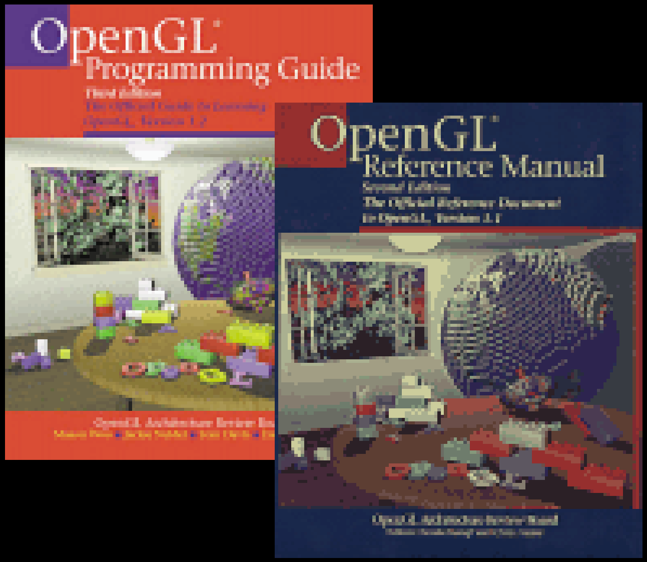
Demo

Summary

- Optimization technique is application dependent
- Understand where your bottle neck is
- Apply available extensions to the problem

For More Information

- In print
    - *OpenGL Programming Guide*
    - *OpenGL Reference Manual*



- On-line

  <http://www.opengl.org>
  <http://lists.apple.com>
    - Search for “opengl”

Roadmap

500 Graphics and Imaging Overview	Room A2 **Tue., 10:30am**
503 Exploring the Quartz Compositor	Hall 2 **Tue., 3:30pm**
504 OpenGL: Graphics Programmability	Room A2 **Tue., 5:00pm**
505 OpenGL: Integrated Graphics I	Room J **Wed., 9:00am**

Roadmap

506 OpenGL: Integrated Graphics II	Room J **Wed., 10:30am**
509 ColorSync and Digital Media	Room C **Wed., 5:00pm**
511 Games Solutions: Graphics, Events, and Tidbits	Room C **Thurs., 10:30am**
512 Games Solutions: NetSprocket and OpenPlay	Room C **Thurs., 2:00pm**

Roadmap

513 OpenGL: Advanced 3D	Room J **Thurs., 3:30pm**
514 OpenGL: **Performance and Optimization**	Room J **Thurs., 5:00pm**
516 Graphics and Imaging **Performance Tuning**	Hall 2 **Fri., 3:30pm**
FF018: **Graphics and Imaging**	Room J1 **Fri., 5:00pm**

Who to Contact

Sergio Mello
3D Graphics Technology Manager
sergio@apple.com

Q&A

Sergio Mello
3D Graphics Technology Manager
Worldwide Developer Relations
sergio@apple.com

http://developer.apple.com/wwdc2002/urls.html

WWDC2002

WWDC2002

WWDC2002